Context Free Languages

Huan Long

Shanghai Jiao Tong University

Acknowledgements

Part of the slides comes from a similar course given by Prof. Yijia Chen.

`http://basics.sjtu.edu.cn/~chen/`

Textbook

Introduction to the theory of computation
Michael Sipser, MIT
Third edition, 2012

Outline

Context free language

Pushdown automata

The pumping lemma for context-free languages

Some decision problems related to PDA

An example

The grammar

$$
\begin{array}{rcl}
A & \rightarrow & 0A1 \\
A & \rightarrow & B \\
B & \rightarrow & \#
\end{array}
$$

A derivation:

$$A \Rightarrow 0A1 \Rightarrow 00A11 \Rightarrow 000A111 \Rightarrow 000\#111.$$

Context-free grammar

Definition

A context-free grammar (CFL) is a 4-tuple (V, Σ, R, S), where

1. V is a finite set called the variables,
2. Σ is a finite set, disjoint from V, called the terminals,
3. R is a finite set of rules, with each rule being a variable and a string of variables and terminals,
4. $S \in V$ is the start variable.

Derivations

Let u, v, w be strings of variables and terminals, and

$$A \to w \quad \in R$$

Then uAv yields uwv: $uAv \Rightarrow uwv$.

u derives v, written $u \stackrel{*}{\Rightarrow} v$, if

- $u = v$, or
- there is a sequence $u_1, u_2, \ldots, u_k$ for $k \geq 0$ and

$$u \Rightarrow u_1 \Rightarrow u_2 \Rightarrow \cdots \Rightarrow u_k \Rightarrow v.$$

The language of the grammar is $\{w \in \Sigma^* \mid S \stackrel{\star}{\Rightarrow} w\}$.

Which is a context-free language(CFL).

Examples

1. Language $\{0^n 1^n \mid n \geq 0\}$, grammar

$$S_1 \to 0S_1 1 \mid \epsilon.$$

2. Language $\{1^n 0^n \mid n \geq 0\}$, grammar

$$S_2 \to 1S_2 0 \mid \epsilon.$$

3. Language $\{0^n 1^n \mid n \geq 0\} \cup \{1^n 0^n \mid n \geq 0\}$, grammar

$$\begin{array}{rcl} S & \to & S_1 \mid S_2 \\ S_1 & \to & 0S_1 1 \mid \epsilon \\ S_2 & \to & 1S_2 0 \mid \epsilon. \end{array}$$

Ambiguity

$$\langle EXPR\rangle \to \langle EXPR\rangle + \langle EXPR\rangle \mid \langle EXPR\rangle \times \langle EXPR\rangle \mid (\langle EXPR\rangle) \mid a$$

The string $a + a \times a$ have two different derivations:

1. $\langle EXPR\rangle \to \langle EXPR\rangle \times \langle EXPR\rangle \Rightarrow \langle EXPR\rangle + \langle EXPR\rangle \times \langle EXPR\rangle \overset{*}{\Rightarrow} a + a \times a.$
2. $\langle EXPR\rangle \to \langle EXPR\rangle + \langle EXPR\rangle \Rightarrow \langle EXPR\rangle + \langle EXPR\rangle \times \langle EXPR\rangle \overset{*}{\Rightarrow} a + a \times a.$

Leftmost derivations

A derivation of a sting w in a grammar G is a leftmost derivation if at every step the leftmost remaining variable is the one replaced.

Ambiguity

Definition

A string w is derived ambiguously is a context free grammar G if it has two or more different leftmost derivations.

Grammar G is ambiguous if it generates some string ambiguously..

$\{a\}$ has two different grammars $S_1 \to S_2 \mid a; S_2 \to a$ and $S \to a$. The first is ambiguous, while the second is not.

$\{a^i b^j c^k \mid i = j \ \vee \ j = k\}$ is inherently ambiguous,i.e., its every grammar is ambiguous.

Ambiguous*

Why care?

Ambiguity of the grammar implies that at least some strings in its language have different structures (parse trees).

1. Thus, such a grammar is unlikely to be useful for a programming language, because two structures for the same string (program) implies two different meanings (executable equivalent programs) for this program.
2. Common example: the easiest grammars for arithmetic expressions are ambiguous and need to be replaced by more complex unambiguous grammars.
3. An inherently ambiguous language would be absolutely unsuitable as a programming language, because we would not have any way of fixing a unique structure for all its programs.

Computational Results *

- There is no algorithm for resolving ambiguity (in the sense of automatically deriving an unambiguous grammar from a given grammar).
- There is not even an algorithm for finding out whether a given CFG is ambiguous.
- However, there are standard techniques for writing an unambiguous grammar that help in most cases.

Chomsy normal form

Definition

A context-free grammar is in Chomsky normal form if every rule is of the form

$$\begin{array}{lcl} A & \rightarrow & BC \\ A & \rightarrow & a \end{array}$$

where a is any terminal and A, B and C are any variables, except that B and C may be not the start variable.

In addition, we permit the rule $S \rightarrow \epsilon$, where S is the start variable.

Theorem

Any context-free language is generated by a context-free grammar in Chomsky normal form.

Proof of the theorem (1)

1. Add a new start variable S_0 with the rule $S_0 \to S$, where S is the original start variable.
2. Remove every $A \to \epsilon$, where $A \neq S_0$.
   For each occurrence of A on the right-hand side of a rule, we add a new rule with that occurrence deleted.
   a) To $R \to uAv$ we add $R \to uv$;
   b) Do the above operation for *each* occurrence of A: e.g. $R \to uAvAw$, we add $R \to uvAw \mid uAvw \mid uvw$.
   c) For $R \to A$, we add $R \to \epsilon$ unless we had previously removed $R \to \epsilon$.
3. Remove every $A \to B$.
   Whenever a rule $B \to u$ appears, where u is a string of variables and terminals, we add the rule $A \to u$ unless this was previously removed.

Proof of the theorem (2)

1. New start variable S_0.
2. Remove every $A \to \epsilon$.
3. Remove every $A \to B$.
4. Replace each rule $A \to u_1 u_2 \cdots u_k$ with $k \geq 3$ and each u_i is a variable or terminal with the rules

   $A \to u_1 A_1, A_1 \to u_2 A_2, A_2 \to u_2 A_3, \cdots,$ and $A_{k-2} \to u_{k-1} u_k.$

   The A_i's are new variables. We replace any terminal u_i with the new variable U_i and add $U_i \to u_i$.

Theorem

If G is a context-free grammar in Chomsky normal form then any $w \in L(G)$ such that $w \neq \epsilon$ can be derived from the start state in exactly $2|w| - 1$ steps.

Proof.

$\square$

Pushdown automata

Definition

A pushdown automata (PDA) is a 6-tuple $(Q, \Sigma, \Gamma, \delta, q_0, F)$, where

1. Q is a finite set of states,
2. Σ is a finite set of input alphabet,
3. Γ is a finite set of stack alphabet,
4. $\delta : Q \times \Sigma_\epsilon \times \Gamma_\epsilon \to \mathcal{P}(Q \times \Gamma_\epsilon)$ is the transition function,
5. $q_0 \in Q$ is the start state,
6. $F \subseteq Q$ is the set of accept states.

Formal definition of computation

Let $M = (Q, \Sigma, \Gamma, \delta, q_0, F)$ be a pushdown automata. M accepts input w if w can be written as $w = w_1 \dots w_m$, where each $w_i \in \Sigma_\epsilon$ and sequences of states $r_0, r_1, \dots, r_m \in Q$ and strings $s_0, s_1, \dots, s_m \in \Gamma^*$ exist that satisfy the following three conditions.

1. $r_0 = q_0$ and $s_0 = \epsilon$.
2. For $i = 0, \dots, m-1$, we have $(r_{i+1}, b) \in \delta(r_i, w_{i+1}, a)$, where $s_i = at$ and $s_{i+1} = bt$ for some $a, b \in \Gamma_\epsilon$ and $t \in \Gamma^*$.
3. $r_m \in F$.

PDA for $\{0^n1^n \mid n \geq 0\}$

$$
\begin{array}{lcl}
Q & = & \{q_1, q_2, q_3, q_4\}, \\
\Sigma & = & \{0, 1\}, \\
\Gamma & = & \{0, \$\}, \\
q_1 & & \text{is the start state} \\
F & = & \{q_1, q_4\}
\end{array}
$$

The transition function is defined by the following table, wherein blank entries signify $\emptyset$

Input:	0			1			ϵ		
Stack:	0	\$	ϵ	0	\$	ϵ	0	\$	ϵ
q_1									$\{(q_2, \$)\}$
q_2			$\{(q_2, 0)\}$	$\{(q_3, \epsilon)\}$					
q_3				$\{(q_3, \epsilon)\}$				$\{(q_4, \epsilon)\}$	
q_4									

Theorem

A language is context free if and only if some pushdown automaton recognizes it.

Proof.

(Only if). Let $G = (V, \Sigma, R, S)$ be a CFL.

1. Place the marker symbol $\$$ and the S on the stack.
2. Repeat the following steps:
   2.1 If the top of stack is some $A \in V$, nondeterministically select some $A \to \omega \in R$ by pushing the string ω on the stack.
   2.2 If the top of stack is some $a \in \Sigma$, read the next symbol from the input and compare it to a. If they match, repeat. Otherwise, reject on this branch of the nondeterminism.
   2.3 If the top of stack is the symbol $\$$, enter the accept state. Doing so accepts the input if it has all been read.

$\square$

Theorem

A language is context free if and only if some pushdown automaton recognizes it.

Proof.

(If).

Simplified PDA:

- It has a single accept state $\{q_{\text{accept}}\}$.
- It empties its stack before accepting.
- Each transition either pushes a symbol onto the stack, or pops one off the stack, but it does not do both at the same time.

Claim

Every PDA has an equivalent simplified PDA.

□

Theorem

A language is context free if and only if some pushdown automaton recognizes it.

Proof.

(If). Give $(Q, \Sigma, \Gamma, \delta, q_0, \{q_{\text{accept}}\})$. We construct CFL G.

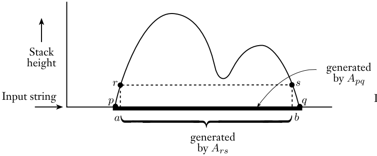


Figure: $A_{pq} \to aA_{rs}b$

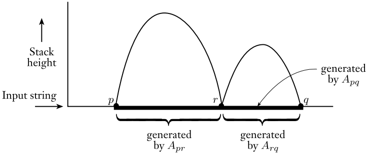


Figure: $A_{pq} \to A_{pr}A_{rq}$

$\square$

Theorem

A language is context free if and only if some pushdown automaton recognizes it.

Proof.

(If). Give $(Q, \Sigma, \Gamma, \delta, q_0, \{q_{\text{accept}}\})$. We construct CFL G. with variables set $\{A_{pq} \mid p, q \in Q\}$, start variable $A_{q_0, q_{\text{accept}}}$. The rules are as followings:

1. For each $p, q, r, s \in Q$, $u \in \Gamma$, and $a, b \in \Sigma_\epsilon$, if $(r, u) \in \delta(p, a, \epsilon)$ and $(q, \epsilon) \in \delta(s, b, u)$, put the rule $A_{pq} \to aA_{rs}b$ in G.
2. For each $p, q, r \in Q$, put the rule $A_{pq} \to A_{pr}A_{rq}$ in G.
3. Finally, for each $p \in Q$, put the rule $A_{pp} \to \epsilon$ in G.

□

Theorem

A language is context free if and only if some pushdown automaton recognizes it.

Claim

If A_{pq} generates x, then x can bring PDA P from state p with empty stack to state q with empty stack.

Claim

If x can bring PDA P from state p with empty stack to state q with empty stack, A_{pq} generates x.

Closure Properties

Theorem

The context-free languages are closed under union, concatenation, and kleene star.

Closure Properties - Union

Proof.

$N_1 = (V_1, \Sigma_1, R_1, S_1)$ recognize A_1,
$N_2 = (V_2, \Sigma_2, R_2, S_2)$ recognize A_2. w.l.o.g. $V_1 \cap V_2 = \emptyset$.

- **Union.** S is a new symbol. Let
  $N = (V_1 \cup V_2 \cup \{S\}, \Sigma_1 \cup \Sigma_2, R, S)$, where
  $R = R_1 \cup R_2 \cup \{S \to S_1, S \to S_2\}$.

$\square$

Closure Properties - Concatenation

Proof.

$N_1 = (V_1, \Sigma_1, R_1, S_1)$ recognize A_1,
$N_2 = (V_2, \Sigma_2, R_2, S_2)$ recognize A_2. w.l.o.g. $V_1 \cap V_2 = \emptyset$.

- **Concatenation.** S is a new symbol. Let $N = (V_1 \cup V_2 \cup \{S\}, \Sigma_1 \cup \Sigma_2, R, S)$, where $R = R_1 \cup R_2 \cup \{S \to S_1 S_2\}$.

$\square$

Closure Properties - Kleene Star

Proof.

$N_1 = (V_1, \Sigma_1, R_1, S_1)$ recognize A_1.

- **Kleene Star.** S is a new symbol. Let $N = (V_1 \cup \{S\}, \Sigma_1, R, S)$, where $R = R_1 \cup \{S \to \epsilon, S \to SS_1\}$.

□

Theorem

The intersection of a context-fee language with a regular language is a context-free language.

Proof.

PDA $M_1 = (Q_1, \Sigma, \Gamma_1, \delta_1, s_1, F_1)$ and
DFA $M_2 = (Q_2, \Sigma, \delta_2, s_2, F_2)$.
Build $M = (Q, \Sigma, \Gamma_1, \Delta, s, F)$, where

- $Q = Q_1 \times Q_2$;
- $s = (s_1, s_2)$;
- $F = (F_1, F_2)$, and
- Δ is defined as follows
  1. for each PDA rule $(q_1, a, \beta) \to (p_1, r)$ and each $q_2 \in Q_2$ add the following rule to Δ
     $$((q_1, q_2), a, \beta) \to ((p_1, \delta_2(q_2, a)), r)$$
  2. for each PDA rule $(q_1, \epsilon, \beta) \to (p_1, r)$ and each $q_2 \in Q_2$ add the following rule to Δ
     $$((q_1, q_2), \epsilon, \beta) \to ((p_1, q_2), r)$$

The pumping lemma for context-free languages

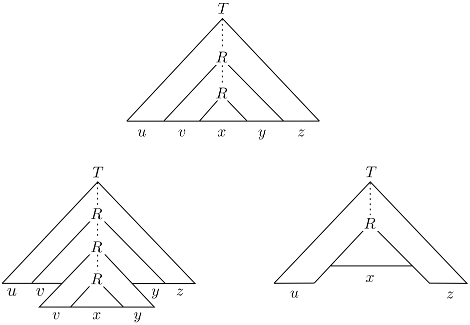
Lemma

If A is a context-free language, then there is a number p (the pumping length) where, if s is any string in A of length at least p, then s may be divided as $s = uvxyz$ satisfying the conditions

1. for each $i \geq 0$, $uv^ixy^iz \in A$,
2. $|vy| > 0$,
3. $|vxy| \leq p$.

Proof (1)

Let G be a CFG for CFL A. Let b be the maximum number of symbols in the right-hand side of a rule. In any parse tree using this grammar, every node can have no more than b children.



Proof (2)

Let G be a CFG for CFL A. Let b be the maximum number of symbols in the right-hand side of a rule. In any parse tree using this grammar, every node can have no more than b children. So, if the height of the parse tree is at most h, the length of the string generated is at most b^h. Conversely, if a generated string is at least $b^h + 1$ long, each of its parse trees must be at least $h + 1$ high.

We choose the pumping length

$$p = b^{|V|+1}$$

For any string $s \in A$ with $|s| \geq p$, any of its parse trees must be at least $|V| + 1$ high.

Proof (3)

Let τ be one parse tree of s with smallest number of nodes, whose height is at least $|V|+1$. So τ has a path from the root to a leaf of length $|V|+1$ with $|V|+2$ nodes. One variable R must appear at least twice in the last $|V|+1$ variable nodes on this path.

We divide s into $uvxyz$:

- u from the leftmost leaf of τ to the leaf left next to the leftmost leaf of the subtree hanging on the first R,
- v from the leftmost leaf of the subtree hanging on the first R to the leaf left next to the leftmost leaf of the subtree hanging on the second R,
- x for all the leaves of the subtree hanging on the second R,
- y from the leaf right next to the rightmost leaf of the subtree hanging on the second R to the rightmost leaf of the subtree hanging on the first R,
- z from the leaf right next to the rightmost leaf of the subtree hanging on the first R to the rightmost leaf of τ.

Proof (4)

Condition 1. Replace the subtree of the second R by the subtree of the first R would validate that for each $i \geq 0$, $uv^ixy^iz \in A$.

Condition 2. If $|vy| = 0$, i.e., $v = y = \epsilon$, then τ cannot have the smallest number of nodes.

Condition 3. To see $|vxy| \leq p = b^{|V|+1}$, note that vxy is generated by the first R. We can always choose R so that its last two occurrences fall within the bottom $|V| + 1$ high. A tree of this height can generate a string of length at most $b^{|V|+1} = p$.

Example

$\{a^n b^n c^n \mid n \geq 0\}$ is not context free.

Proof.

Assume otherwise, and let p be the pumping length. Consider $s = a^p b^p c^p$ and divide it to $uvxyz$ according to the Pumping Lemma.

- When both v and y contain only one type of symbols, i.e., one of a, b, c, then uv^2xy^2z cannot contain equal number of a's, b's, and c's.
- If either v or y contains more than one type of symbols, then uv^2xy^2z would have symbols interleaved.

□

Example

$\{ww \mid w \in \{0,1\}^*\}$ is not context free.

Proof.

Assume otherwise, and let p be the pumping length. Consider $s = 0^p 1^p 0^p 1^p$ and divide it to $uvxyz$ with $|vxy| \leq p$.

- If vxy occurs only in the first half of s, then the second half of uv^2xy^2z must start with an 1. This is impossible
- Similarly vxy cannot occur only in the second half of s.
- If vxy straddles the midpoint of s, then pumping s to the form $0^p 1^i 0^j 1^p$ cannot ensure $i = j = p$.

$\square$

Theorem

The context free language are not closed under intersection or complementation.

Proof.

Clearly $\{a^n b^n c^m \mid m, n \geq 0\}$ and $\{a^m b^n c^n \mid m, n \geq 0\}$ are both CFL. However their intersection, $\{a^n b^n c^n \mid n \geq 0\}$, is not.

To the second part of the statement,

$$L_1 \cap L_2 = \overline{\overline{L_1} \cup \overline{L_2}}$$

rules out the closure under complementation. □

Language	regular	context-free
Machine	DFA/NFA	PDA
Syntax	regular expression	context-free grammar

Problems from formal language theory

Decision Problems

- Acceptance: does a given string belong to a given language?
- Emptiness: is a given language empty?
- Equality: are given two languages equal?

Language Problems concerning CFL

Theorem

The following three problems:

- Acceptance: Given a CFG G and a string w, does G accept w?
- Emptiness: Given a CFG G is the language $L(G)$ empty?
- Equality: Given two CFG A and B is $L(A)$ equal to $L(B)$?

The Acceptance and Emptiness problem for CFG are decidable, the Equality problem is not decidable.